Appendix A: Example of Server Executable Code In Recognition/Annotation Process

bl_annotate.pl

```
#!/usr/bin/perl -w

######
# Script to annotate the buylites
# Written by Henri
#
######

use strict;
use LWP::UserAgent;
use Apache::Registry;
use Time::HiRes qw (time);

########## Pull in words from page and compare to hash tree ###########
my $url;
my $url_unescape;
my $textdata;
my @tempmatchingids=();       # temp array to return longest matching
word
my %finallist=();       # hash that holds the final matching proper
nouns as its keys
my $i=0;
my $j=0;
my $res='';
my @words;
my $max_annotations = 15; # steve - 5.24.01

$url = $ENV{'QUERY_STRING'};
$url =~ s/(^|&)ref=([^&]+).*$/$2/o;
$url_unescape = $url;
$url_unescape =~ s/%([0-9A-Fa-f]{2})/chr(hex($1))/eg; # Unescape the
stuff

############ Get the Page #########
# Need to use LWP::UserAgent instead of LWP::Simple because we need the
timeout function
my $ua = new LWP::UserAgent;
$ua->timeout(10); # 10 second timeout
my $request = new HTTP::Request('GET', $url_unescape);
$textdata = $ua->request($request);
if ($textdata->is_success) {
      $textdata = $textdata->content;
```

```
} else {
      $textdata = '';
}
######### Done w/ the Page ########
my $aa = time;

# $textdata =~ s/<IMG[^>]+>//igo;
$textdata =~ s/<[A|a] .+?\/[A|a]>//go;
$textdata =~ s/<SCRIPT.+?<\/SCRIPT>//igo;
$textdata =~ s/<HEAD( |>).+?\/HEAD>//igo;
# $textdata =~ s/<FORM.+?<\/FORM>//igo;
# $textdata =~ s/<object.+?<\/object>//igo;
# $textdata =~ s/<embed.+?<\/embed>//igo;
$textdata =~ s/<[^>]+>//go;
$textdata =~ s/&[^;]+;//go;
# $textdata =~ s/\[[^]]+]//igo;

@words = split /[^a-zA-Z0-9\-'\/]+/, $textdata;

my $wordcount = scalar (@words);
my $numlist = 0;
my $stringtosearch ='';
my $nounid;
my $maxwords = 10;      # maximum number of words in one buylite string

######### BEGIN RECOGNITION ############
while (($i < $wordcount) && ($numlist < $max_annotations)) {

      # Throw out the 1st word if it's not in the first word list
      # This is only to accelerate dramatically the searching
      if (not exists ($ANNOTATOR::firstword{$words[$i]})) {
            $i++;
            next;
      }

      $j=$i;
      $stringtosearch ='';

      while (($j-$i <= $maxwords) && ($j < $wordcount)) {
# Are we not yet at end of file?
            $stringtosearch .= $words[$j];
            if (exists($ANNOTATOR::pnlist{$stringtosearch})) {
                  push @tempmatchingids, $stringtosearch;
                  $i=$j+1;
            }
            $j++;
            $stringtosearch .= ' ';
      }

      if (scalar(@tempmatchingids) > 0) {             # did we have
earlier full keyphrase matching?
            $finallist{pop @tempmatchingids} = 1;
```

```
                $numlist++;
                @tempmatchingids = ();
        } else {
                $i++;
        }
} # while
############ END RECOGNITION ############

$aa = time - $aa;
print STDERR ("$url_unescape --- $aa\n");

############ Output ############
my $linkstag;
my $cache_header;
my $outfile;

if ($numlist == 0) {          # No buylites found
        $outfile = 'function makeChanges() {}; function blHighlight()
{}';
        goto PRINTOUTPUT;
}

$ENV{'QUERY_STRING'} =~ /(^|&)pn=([^&])/o;
if ((defined $2) && ($2 eq 'n')) {              # buylites disabled
through cookie
        $outfile = 'function makeChanges() {}; function blHighlight() {
blInitUI(BL_HIDE); }';
        goto PRINTOUTPUT;
}
# From here onwards we found buylites and they are enabled on the
client

$outfile = << "EOT";

function makeChanges() { var i9i; var refurl = '$url'; [LINKS_TAG]
}function blHighlight(){TAG_NAME=new
Array();TAG_NAME[0]='P';TAG_NAME[1]='LI';TAG_NAME[2]='UL';TAG_NAME[3]='
TD';TAG_NAME[4]='DL';var pnshown=0;for(var
i=0;i<TAG_NAME.length;i++){var
ps=document.all.tags(TAG_NAME[i]);for(j=0;j<ps.length;j++){g_html=ps(j)
.innerHTML;g_outer=ps(j).outerHTML;g_tag=g_outer.substring(0,g_outer.in
dexOf('>')+1);if((g_html.length>=5000)||(g_html==''))continue;for(var
p=1;p<pnpatbeg.length;p++)g_html=replacepat(g_html,p);chgflag=0;pntmpli
st='';makeChanges();for(var p=pnpatbeg.length-1;p>0;p--
)g_html=expandpat(g_html,p);if(chgflag==1){var
pnrv=1;if(i!=3)pnrv=setOuterHTML(ps(j),g_tag+g_html+'</'+TAG_NAME[i]+'>
');else pnrv=setInnerHTML(ps(j),g_html);if(pnrv==0)pnshown=1;}for(var
p=1;p<pnpatbeg.length;p++)pnpattotal[p]=0;}}if(pnshown==1)blInitUI(BL_S
HOW);}
EOT

foreach $res (keys %finallist) {
```

GANZLAW, PC
PO Box 10105
Portland, Oregon 97296
Phone: (503) 228-3641
Docket No.: BIZ/01-0003
Express Mail No: EL627039307US, Deposited June 12, 2001

```
#     $linkstag = "changeHTML(\"".$ANNOTATOR::pnlist{$res}."\",
\"http://colostage.bizrate.com/buylites/landing.xpml?keyword=".$ANNOTAT
OR::pnlist{$res}."&aff_id=-1&noun_id=".$res."&rf=buy".$res."|-
1&ref_url=\" + refurl);";
      $linkstag = "changeHTML(\"".$res."\",
\"http://colostage.bizrate.com/buylites/landing.xpml?keyword=".$res."&a
ff_id=-
1&noun_id=".$ANNOTATOR::pnlist{$res}."&rf=buy".$ANNOTATOR::pnlist{$res}
."|-1&ref_url=\" + refurl);";
      $outfile =~ s/\[LINKS_TAG]/$linkstag [LINKS_TAG]/;
}
$outfile =~ s/\[LINKS_TAG]//o;

PRINTOUTPUT:
print "HTTP/1.0 200 OK\n";
print "Content-Length: " . length($outfile) . "\n";
print "Cache-Control: max-age=86400\n";
print "Content-type: text/html\n\n";
print $outfile;
```

Appendix B: Example of Client Executable Code In Recognition/ Annotation Process

The following four server-side components are sent by one or more servers to a client for execution in the following order on the client:

buylites.js (JScript file) [Normally found on content document initially executed on a client system]

bl_style.css (stylesheet)

bl_vb.js (VBScript file)

bl_start.js (JScript file)

buylites.js

```
var BL_BURL = 'http://xxx.xxxxx.com/buylites';
var BL_SURL = BL_BURL+'/bl';
var BL_IURL = BL_SURL+'/images';
var blNavAgt = navigator.userAgent.toLowerCase();
var blIsMajor = parseInt(navigator.appVersion);
var blIsIE = (blNavAgt.indexOf("msie") != -1);
var blIsIE4 = (blNavAgt.indexOf("msie 4") != -1);
var blIsIE4up = (blIsIE && (blIsMajor >= 4));
var blIsWin = (blNavAgt.indexOf("windows ")!=-1);

if (blIsIE4up && blIsWin)
{
        if (typeof(BL_AFF_ID) == 'undefined') BL_AFF_ID = -1;
        document.write('\
        <link href="'+BL_SURL+'/bl_style.css" rel=stylesheet type="text/css">\
        <script language=jscript src="'+BL_SURL+'/bl_main.js"></script>\
        <script language=VBScript src="'+BL_SURL+'/bl_vb.js"></script>\
        <script                                                language=jscript
src="'+BL_BURL+'/r1.pl?aff_id='+BL_AFF_ID+'"></script>\
        <script language=jscript src="'+BL_SURL+'/bl_start.js"></script>');
}
```

GANZLAW, PC
PO Box 10105
Portland, Oregon 97296
Phone: (503) 228-3641
Docket No.: BIZ/01-0003
Express Mail No: EL627039307US, Deposited June 12, 2001

bl_style.css

```
.boxtext1 {color:#000197;font-family: Arial, Helvetica, sans-serif; font-size: 12;
font-weight: bold;}
.boxtext2 {color:#FF6600;font-family: Verdana, Arial, Helvetica, sans-serif; font-
size: 16; font-weight: bold;}
.boxtext3 {color:#000000;font-family: Verdana, Arial, Helvetica, sans-serif; font-
size: 13; font-weight: bold;}
.boxtext4 {color:blue}
.boxtext5 {color:#000197;font-family: Arial, Helvetica, sans-serif; font-size: 11;}
.boxtext6 {color:black;font-family: Arial, Helvetica, sans-serif; font-size: 12;}
.boxtext7 {color:black;font-family: Arial, Helvetica, sans-serif; font-size: 11;}
.pnlg                    {background-color=#fcff00;BACKGROUND-IMAGE:
url(/buylites/bl/images/cornerg1.gif);BACKGROUND-POSITION:left
top;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.pncg {background-color=#fcff00;cursor:hand}
.pnrg    {text-decoration:none;background-color=#fcff00;BACKGROUND-IMAGE:
url(/buylites/bl/images/cornerg2.gif);BACKGROUND-POSITION:right
bottom;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.pnllg                   {background-color=#fcff00;BACKGROUND-IMAGE:
url(/buylites/bl/images/cornerlg1.gif);BACKGROUND-POSITION:left
top;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.pnclg {background-color=#fcff00;cursor:hand}
.pnrlg   {text-decoration:none;background-color=#fcff00;BACKGROUND-IMAGE:
url(/buylites/bl/images/cornerlg2.gif);BACKGROUND-POSITION:right
bottom;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.pnl1          {color:#000000;background-color=#fcff00;BACKGROUND-IMAGE:
url(/buylites/bl/images/corner1.gif);BACKGROUND-POSITION:left
top;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.pnc1 {color:#000000;background-color=#fcff00;cursor:hand}
.pnr1          {color:#000000;background-color=#fcff00;BACKGROUND-IMAGE:
url(/buylites/bl/images/corner2.gif);BACKGROUND-POSITION:right
bottom;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.l2                      {background-color=#11fff1;BACKGROUND-IMAGE:
url(/buylites/bl/images/corner1.gif);BACKGROUND-POSITION:left
top;BACKGROUND-REPEAT:no-repeat;}
.c2   {background-image:   url(/buylites/bl/images/pattern1blue.gif);background-
position: left bottom;background-repeat:repeat-x}
.r2                                                  {BACKGROUND-IMAGE:
url(/buylites/bl/images/pattern1_dotblue.gif);BACKGROUND-POSITION:center
top;BACKGROUND-REPEAT:no-repeat;}
```

GANZLAW, PC
PO Box 10105
Portland, Oregon 97296
Phone: (503) 228-3641

```
.l3                    {background-color=#11fff1;BACKGROUND-IMAGE:
url(/buylites/bl/images/corner1.gif);BACKGROUND-POSITION:left
top;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.c3 {background-color=#11fff1;cursor:hand}
.r3     {text-decoration:none;background-color=#11fff1;BACKGROUND-IMAGE:
url(/buylites/bl/images/corner2.gif);BACKGROUND-POSITION:right
bottom;BACKGROUND-REPEAT:no-repeat;cursor:hand}
.pnspan                {text-decoration:none;color=#000099;background-
color=#CCCCCC;BACKGROUND-IMAGE:
url(http://images.bizrate.com/site/un_b_bold.gif);BACKGROUND-POSITION:left
bottom;BACKGROUND-REPEAT:repeat-x;}
.pnspan2 {background-color=#11fff1; cursor:hand}
.pnspan3 {background-color=#11fff1;}
.pnul    {text-decoration:none;    color:#000000;    background-color=#fcff00;
BACKGROUND-IMAGE:
url(/buylites/bl/images/underline_buylites.gif);BACKGROUND-POSITION:left
bottom; BACKGROUND-REPEAT:repeat-x; cursor:hand;}
```

bl_vb.js

```
Function setInnerHTML(elem, val)
      ON ERROR RESUME NEXT
      elem.innerHTML = val
        setInnerHTML = Err.Number
END Function

Function setOuterHTML(elem, val)
      ON ERROR RESUME NEXT
      elem.outerHTML = val
        setOuterHTML = Err.Number
END Function
```

bl_start.js

```
var blTry = 0;

function blShowBLs()
{
      if (typeof(blHighlight) == 'function' && typeof(makeChanges) == 'function')
      {
            blHighlight();
```

```
        }
        else
        {
                blTry++;
                if (blTry < 10)
                        setTimeout('blShowBLs();', BL_DELAY_RETRY);
        }
}

window.onload = blShowBLs;

© BizRate.com
```

Appendix C: Example of Client-Side Annotation and Recognition Code

Defiler.cpp

GnomeIEPageBroker.cpp

HTMLAnnotatorProxy.cpp

HTMLParser.cpp

IEDocUtils.cpp

ProperNounDB.cpp

Defiler.cpp

```
#include <stdafx.h>
#include "Defiler.h"
#include "HTMLParser.h"
#include "ProperNounDB.h"
#include "bizratestring.h"
#include <sstream>
#include "ebRegistryMgr.h"
#include "eBBarRegistryMgr.h"
#include "eBconst.h"
#include "eBScriptUtil.h"
#include "IEDocUtils.h"

using namespace std;

const int Defiler::AFFILIATE_ID = -2;

Defiler::Defiler(ProperNounTable *pnTable,const string &serverBaseURL) {
        this->pnTable=pnTable;                  // This object takes ownership of the
pnTable.


  eBRegistryMgr regMgrLM;
  regMgrLM.InitAppKey( HKEY_LOCAL_MACHINE, eb_eboodle );

  // Get Version servlet
  CComBSTR redirectServlet;
  regMgrLM.GetProfileString(RS_PRPOERNOUN,      RS_PN_REDIRECT_SERVLET,
&redirectServlet );
```

```
    if (redirectServlet.Length() == 0)
    {
       WriteLog(EBLOG_OBJECT, 0, "Defiler::Defiler: Invalid redirect servlet\n");
            return;
    }
    _bstr_t bstrRedirectServlet = redirectServlet.m_str;
          annotationServletURL=serverBaseURL+((char *)bstrRedirectServlet);
}

Defiler::~Defiler() {
          delete pnTable;
}

string *Defiler::AnnotateHTML(const string *originalHTML, IHTMLElement *elem,
CComPtr<IWebBrowser2> pIWebBrowser, IDispatch* pDisp, bool &bAddSpanTag) {
          string *annotatedHTML = NULL;
          AnnotationResult retval=UNCHANGED;
          HTMLParser parser(*originalHTML);
          const DocumentWordList *docWords=parser.GetDocumentWords();

          ANNOTATION_SET *annSet=GetAllAnnotations(docWords,originalHTML);
          HRESULT hinsertedCode;
          if(annSet->size()>0)
          {
                    if (!bAddSpanTag)
                    {
                   CComPtr<IWebBrowser2> pMainWebBrowser;
                        pDisp->QueryInterface(IID_IWebBrowser2,(void **)&pMainWebBrowser);
                                  hinsertedCode    =    InsertProperNounScript(pIWebBrowser,
pMainWebBrowser);
                              if(hinsertedCode == S_OK)
                                    bAddSpanTag = true;
                    }
                    if(bAddSpanTag)
                                  annotatedHTML=InsertPopup(annSet,originalHTML,          elem,
pIWebBrowser, pDisp);
          }

          delete annSet;
          return annotatedHTML;
}

HRESULT  Defiler::InsertProperNounScript(CComPtr<IWebBrowser2>  pIWebBrowser,
IDispatch* pDisp)
```

```
{
        CComPtr<IWebBrowser2> pmainWebBrowser;
        if(pDisp)
                pDisp->QueryInterface(IID_IWebBrowser2,(void **)&pmainWebBrowser);
        CComPtr<IHTMLDocument2> pdoc;
        if(pmainWebBrowser)
        {
                CComPtr<IDispatch> pdispDoc;
                pmainWebBrowser->get_Document(&pdispDoc);
                pdispDoc->QueryInterface(IID_IHTMLDocument2,(void **)&pdoc);
        }
        if(!pdoc)
                return S_FALSE;
  CComVariant variantList[1];
  variantList[0].vt = VT_DISPATCH;
  pdoc->QueryInterface(IID_IDispatch,(void **)&(variantList[0].pdispVal));
  CComVariant vResult;
  eBScript::ExecFunc( pIWebBrowser, L"popupx", 1, variantList, &vResult );
  if (vResult.vt == VT_I4 && vResult.intVal == 1)
                return S_OK;
  else
        return S_FALSE;

}

string *Defiler::InsertPopup(ANNOTATION_SET *annSet,const string *originalHTML,
                        IHTMLElement *elem, CComPtr<IWebBrowser2> pIWebBrowser,
IDispatch* pDisp) {
        ostringstream oss;

        ANNOTATION_SET::const_iterator iter=annSet->begin();
        ANNOTATION_SET::const_iterator theEnd=annSet->end();

        int nextPlaceToCopy=0;

        while(iter != theEnd) {
                const AnnotationWrapper &wrapper=*iter;
                const Annotation *annotation=wrapper.GetAnnotation();
                int startPos=annotation->GetStartIndex();
                int endPos=annotation->GetEndIndex();

                string htmlPN   = originalHTML->substr(startPos,endPos-startPos);

                string  htmlStart  =  originalHTML->substr(nextPlaceToCopy,startPos-
nextPlaceToCopy);

                oss << htmlStart;
```

```
                string *annotationURL=GenerateAnnotationURL(*annotation,htmlPN);

                CComBSTR bstrSpan;
        GetSpanHTML(pIWebBrowser, pDisp, htmlPN, *annotationURL, bstrSpan);

                if (bstrSpan.Length() > 0)
                {
          _bstr_t tmpSpan = bstrSpan.m_str;
          oss << (char *)tmpSpan;
                }

            //string outputString = string(oss.str());
      //WriteLog(EBLOG_OBJECT,    0,    "Defiler::InsertPopup:    All_HTML=%s\n",
outputString.c_str());

                nextPlaceToCopy=endPos;
                iter++;
          }

          oss << originalHTML->substr(nextPlaceToCopy);

          //string outputString = string(oss.str());
    //WriteLog(EBLOG_OBJECT,    0,    "Defiler::InsertPopup:    All_HTML=%s\n",
outputString.c_str());

          string annotatedHTML = oss.str();
          IEDocUtils::SetHTML(elem, annotatedHTML.c_str());

          return new string(oss.str());
}

HRESULT Defiler::GetSpanHTML(CComPtr<IWebBrowser2> pIWebBrowser, IDispatch*
pDisp,
                                const string &strPN, const string &urlPN, CComBSTR
&bstrSpan)

{
          int retval=0;

    CComVariant variantList[3];

    CComBSTR bstrURL = urlPN.c_str();
    variantList[0].vt = VT_BSTR;
    variantList[0].bstrVal = bstrURL.Copy();

    CComBSTR bstrPN = strPN.c_str();
    variantList[1].vt = VT_BSTR;
```

```
    variantList[1].bstrVal = bstrPN.Copy();

    variantList[2].vt = VT_DISPATCH;
    pDisp->QueryInterface(IID_IDispatch,(void **)&(variantList[2].pdispVal));

        CComVariant vResult ;
    eBScript::ExecFunc( pIWebBrowser, L"getSpanTag", 3, variantList, &vResult );


        if (vResult.vt == VT_BSTR)
        {
      bstrSpan = vResult.bstrVal;
          return S_OK;
        }
        else
                return S_FALSE;
}


ANNOTATION_SET        *Defiler::GetAllAnnotations(const        DocumentWordList
*docWords,const string *originalHTML) {
        ANNOTATION_SET *set=new ANNOTATION_SET;

        unsigned int numWords=docWords->NumWords();
        unsigned int currentWord=0;

        while(currentWord<numWords) {
                ProperNounTable *curTable=pnTable;
                const DocumentWord *firstWord=docWords->GetWord(currentWord);
                const DocumentWord *lastWord=firstWord;

                char firstChar=originalHTML->at(firstWord->GetStringIndex());
                if( (::isUpperCaseLetter(firstChar)) || (::isNumber(firstChar)) ) {
                        // Only check for annotations if the first character is
                        // upper case or a number.
                        while(true) {
                                if(currentWord>=numWords)
                                        break;

                                const        DocumentWord        *tmpWord=docWords-
>GetWord(currentWord);
                                const string *word=tmpWord->GetWord();

                                ProperNounTable                *nextTable=curTable-
>GetTableForWord(*word,false);
                                if(nextTable==NULL)
                                        break;
```

```
                                curTable=nextTable;
                                lastWord=tmpWord;
                                currentWord++;
                        }
                }

                if(curTable==pnTable) {
                        // Never annotate at the root table.  Skip over this word
                        // becayse it matched nothing.
                        currentWord++;
                } else {
                        // Okay, so some words matched.  See if there is a proper noun
                        // associated with the last table that contained a proper noun
                        // word.
                        const ProperNoun *properNoun=curTable->GetProperNoun();
                        if(properNoun!=NULL) {
                                // Found a proper noun match.  Add the annotation.
                                Annotation annotation;
                                annotation.SetAssociatedProperNoun(properNoun);
                                annotation.SetStartIndex(firstWord->GetStringIndex());
                                annotation.SetEndIndex(lastWord-
>GetStringIndex()+lastWord->GetWord()->size());
                                set->insert(AnnotationWrapper(&annotation));
                        }
                }
        }

        return set;
}



string *Defiler::GenerateAnnotationURL(const Annotation &annotation,string Keywords )
{
        const ProperNoun *pn=annotation.GetAssociatedProperNoun();
        ostringstream oss;

        oss << annotationServletURL <<"?keyword="<<Keywords<< "&aff_id=" <<
AFFILIATE_ID
                << "&noun_id=" << pn->GetId() << "&fb=1&rf=bar" <<pn->GetId() << "-
aff_id=" ;

        return new string(oss.str());
}
```

GnomeIEPageBroker.cpp

```
#include <stdafx.h>
#include "GnomeIEPageBroker.h"
#include "IEDocUtils.h"
#include "GnomeConfig.h"
#include "ListLoader.h"
#include <string>
#include "eBScriptUtil.h"

using namespace std;


////Defiler *GnomeIEPageBroker::defiler = NULL;

////int GnomeIEPageBroker::defilerRefCount = 0;

//BRLock *GnomeIEPageBroker::refCountLock = LockFactory::MakeLock();


GnomeIEPageBroker::GnomeIEPageBroker(GnomeBaseConfig *config, int
pnListServerVersion) {
        // This object takes ownership of config.
        defiler=NULL;
        this->config=config;
        Init(pnListServerVersion);

}


GnomeIEPageBroker::~GnomeIEPageBroker() {
        delete annotatorProxy;
        delete config;

        // Decrease the reference count on the defiler.  If the count is
        // zero, delete it; otherwise, keep it around because other people
        // are using it.
        /*refCountLock->Lock();

        defilerRefCount--;
        if(defilerRefCount==0) {
        */
                if(defiler)
                        delete defiler;
                defiler=NULL;
```

```
        ////}

        ////refCountLock->Unlock();
    }

int GnomeIEPageBroker::ProcessAllFrames(CComPtr<IWebBrowser2> pIWebBrowser,
IDispatch * pDisp) {
        if( (defiler == NULL) || !pDisp )
        {
           WriteLog("GnomeIEPageBroker::ProcessAllFrames: annotations off\n");
               return 0;                        // Disabled annotations.
        }
        CComPtr<IWebBrowser2> pmainWebBrowser;
        if(pDisp)
                pDisp->QueryInterface(IID_IWebBrowser2,(void **)&pmainWebBrowser);
        CComPtr<IHTMLDocument2> pdoc;
        if(pmainWebBrowser)
        {
                CComPtr<IDispatch> pdispDoc;
                pmainWebBrowser->get_Document(&pdispDoc);
                pdispDoc->QueryInterface(IID_IHTMLDocument2,(void **)&pdoc);
        }
        if(!pdoc)
                return 0;
        CComVariant variantList[1];
        variantList[0].vt = VT_DISPATCH;
        pdoc->QueryInterface(IID_IDispatch,(void **)&(variantList[0].pdispVal));
        CComVariant vResult;

        eBScript::ExecFunc( pIWebBrowser, L"isValidtoAnnotate", 1, variantList,
&vResult );
        if (vResult.vt == VT_I4 && vResult.intVal == 1)
        {
                ProcessPage(pdoc,true, pDisp, pIWebBrowser);
        }
        return 0;
}
int GnomeIEPageBroker::ProcessPage(IHTMLDocument2 *doc,bool updateURLList,
IDispatch* pDisp, CComPtr<IWebBrowser2> pIWebBrowser) {
        // Annotate a single page.
        WriteLog("GnomeIEPageBroker::ProcessPage: in\n");
        /*if(updateURLList) {
                char *url=IEDocUtils::GetURL(doc);
                urlList.AddURL(url);
                delete[] url;
        }*/
```

```
        // Annotate the document.
        IEDocUtils::WalkThroughDocument(annotatorProxy, doc, pIWebBrowser, pDisp);

        return 0;
}


//bool GnomeIEPageBroker::Enabled() {
//      return config->GetAnnotationsEnabled();
//}


void GnomeIEPageBroker::Init(int pnListServerVersion) {
        annotatorProxy=NULL;

        if(defiler==NULL) {
                // Only do these steps if the defiler has not been set up yet.
                string *serverURLBase=config->GetServerURLBase();
                ListLoader loader(*serverURLBase);
                delete serverURLBase;
        int clientPNListVer = config->GetPNListVersion();
                if(pnListServerVersion > clientPNListVer) {
                        // A newer version is available.
                        // Load the proper noun list from the network.
                        loader.GetProperNounsFromURL(new
AsynchronousLoadCallback(this),pnListServerVersion);
                } else {
                        // A newer version is not available.  Load it from the
                        // filesystem.
                        string *cacheFilename=config->GetCacheFilename();
                        ProperNounTable
*pnTable=loader.GetProperNounsFromFile(*cacheFilename);
                        delete cacheFilename;
                        if(pnTable==NULL) {
                                // Exception.
                                // Something happened to the proper noun list in the
                                // filesystem.  Load it from the network.
                                loader.GetProperNounsFromURL(new
AsynchronousLoadCallback(this),pnListServerVersion);
                        } else {
                                SetDefiler(pnTable);
                        }
                }
        } else {
                SetAnnotatorProxy();
        }
```

GANZLAW, PC
PO Box 10105
Portland, Oregon 97296
Phone: (503) 228-3641

```
}

void GnomeIEPageBroker::SetAnnotatorProxy() {
        delete annotatorProxy;
        annotatorProxy=new HTMLAnnotatorProxy(defiler,config);

        ////refCountLock->Lock();
        ////defilerRefCount++;
        ////refCountLock->Unlock();
}

void GnomeIEPageBroker::SetDefiler(ProperNounTable *pnTable) {
        // This object takes ownership of pnTable.
        delete defiler;
        string *annotationURLBase=config->GetAnnotationURLBase();
        defiler=new Defiler(pnTable,*annotationURLBase);
        delete annotationURLBase;
        ////defilerRefCount=0;

        SetAnnotatorProxy();
}

int GnomeIEPageBroker::TakeLoad(ProperNounTable *pnTable,const std::string
*pnText,unsigned int latestVersion) {
        // Create a new defiler from the proper noun table.  Then, store the
        // latest proper noun list version and the list text in a file.

        // set the latest version in the registry
        eBRegistryMgr regMgrLM;
        regMgrLM.InitAppKey( HKEY_LOCAL_MACHINE, eb_eboodle );
        wchar_t wc[100];
        _itow(  latestVersion , wc, 10);
        regMgrLM.WriteProfileString(RS_PRPOERNOUN,
RS_PN_CLIENT_PN_VERSION,  wc);
        SetDefiler(pnTable);

        string *cacheFilename=config->GetCacheFilename();
        ListLoader::StoreProperNounsInFile(*cacheFilename,*pnText);
        delete cacheFilename;

        return 0;
}
```

HTMLAnnotatorProxy.cpp

```
#include <stdafx.h>
#include "HTMLAnnotatorProxy.h"
#include "IEDocUtils.h"
#include <string>
#include <sstream>
#include "eBScriptUtil.h"

using namespace std;


HTMLAnnotatorProxy::HTMLAnnotatorProxy(Defiler *defiler,GnomeBaseConfig *config)
{
        this->defiler=defiler;
        //RelevantTags::init();

        if(config->InDebugMode()) {
                string *cacheDir=config->GetCacheDirectory();
                debugHTMLWriter=new DebugHTMLWriter(cacheDir->c_str());
                delete cacheDir;
        } else
                debugHTMLWriter=NULL;

        annotationNum=0;
}


HTMLAnnotatorProxy::~HTMLAnnotatorProxy() {
        delete debugHTMLWriter;
}


int HTMLAnnotatorProxy::Process(CComPtr<IWebBrowser2> pIWebBrowser, IDispatch*
pDisp, IHTMLElement *elem, bool &bAddSpanTag)
{

   USES_CONVERSION;


        //CComBSTR tagNameBSTR;
        //HRESULT result=elem->get_tagName(&tagNameBSTR);

        //if(FAILED(result))
        //      return -1;
```

```
        //_bstr_t tagNameWrapper(tagNameBSTR.m_str); //  tagNameWrapper  takes
ownership of tagNameBSTR.

        //if(!RelevantTags::IsRelevantTag(&tagNameWrapper))
        //      return 0;

        int elemCountAllowed = isElementValidity(pIWebBrowser,elem);
        if( !elemCountAllowed )
                return 0;

        //char *origHTML=IEDocUtils::GetHTML(elem); commented by RAJ
        CComBSTR htmlBSTR;
        elem->get_innerHTML(&htmlBSTR);


        if( elemCountAllowed > 0 && elemCountAllowed != 1 )
        {
                if(htmlBSTR.m_str && (wcslen(htmlBSTR.m_str) > elemCountAllowed))
                        return 0;
        }

        if( (htmlBSTR.m_str != NULL)  ) {
        //if( (htmlBSTR.m_str != NULL) && (wcslen(htmlBSTR.m_str)<4000) ) {
                string                                          *annotatedHTML=defiler-
>AnnotateHTML(&string(W2A(htmlBSTR.m_str)),     elem,     pIWebBrowser,     pDisp,
bAddSpanTag);

                if(annotatedHTML != NULL) {
                        // There is an annotation, so do it.
                        if(debugHTMLWriter != NULL) {
                                // Write out the changes to a file.
                                string *filePrefix=GetHTMLWriterFilename();
                                const char *filePrefixCStr=filePrefix->c_str();
                                //debugHTMLWriter-
>WriteHTML(filePrefixCStr,origHTML,ORIGINAL_HTML);
                                debugHTMLWriter-
>WriteHTML(filePrefixCStr,annotatedHTML->c_str(),ANNOTATED_HTML);
                                delete filePrefix;
                        }

//              WriteLog("HTMLAnnotatorProxy::Process: Before IEDocUtils::SetHTML\n");
//       SetSpanHTML(pIWebBrowser, pDisp, elem,annotatedHTML->c_str());
//                      IEDocUtils::SetHTML(elem,annotatedHTML->c_str());
                        delete annotatedHTML;
                }
        }
```

```
        //delete[] origHTML;

        return 0;
}

int                 HTMLAnnotatorProxy::isElementValidity(CComPtr<IWebBrowser2>&
pIWebBrowser,IHTMLElement * elem)
{
        if(!elem)
                return 0;

        CComVariant variantRet;
    CComVariant variantList[1];
    variantList[0].vt = VT_DISPATCH;
    elem->QueryInterface(IID_IDispatch,(void **)&(variantList[0].pdispVal));
    eBScript::ExecFunc( pIWebBrowser, L"isValidElement", 1, variantList, &variantRet );
        if(variantRet.vt == VT_I4 && variantRet.lVal == 1 )
                return 1;
        else if(variantRet.vt == VT_I4 && variantRet.lVal > 1 )
                return variantRet.lVal ;
        else
                return 0;
}

int HTMLAnnotatorProxy::SetSpanHTML(CComPtr<IWebBrowser2> pIWebBrowser,
                                                        IDispatch*      pDisp,
IHTMLElement *elem,const char *html) {
        int retval=0;
    CComVariant variantList[3];

    CComBSTR strHTML = html;
    variantList[0].vt = VT_BSTR;
    variantList[0].bstrVal = strHTML.Copy();

    variantList[1].vt = VT_DISPATCH;
    elem->QueryInterface(IID_IDispatch,(void **)&(variantList[1].pdispVal));

    variantList[2].vt = VT_DISPATCH;
    pDisp->QueryInterface(IID_IDispatch,(void **)&(variantList[2].pdispVal));

    eBScript::ExecFunc( pIWebBrowser, L"addSpanTag", 3, variantList, NULL );
        return retval;
}

void HTMLAnnotatorProxy::SetDefiler(Defiler *defiler) {
        this->defiler=defiler;
}
```

```
string *HTMLAnnotatorProxy::GetHTMLWriterFilename() {
        ostringstream oss;
        oss << "X_" << (annotationNum++);
        return new string(oss.str());
}



/************************************************************
 *                                          *
 * RelevantTags implementation.                    *
 *                                          *
 ************************************************************/

/*
int RelevantTags::init() {
        // Initialize the static members of the class if they have not already
        // been initialized.
        if(!isInitialized) {
                isInitialized=true;
                AddRelevantTags();
                return 1;
        }

        return 0;
}


bool RelevantTags::IsRelevantTag(const _bstr_t *tagWrapper) {
        return (tagset.find(*tagWrapper) != tagset.end());
}


void RelevantTags::AddRelevantTags() {
        // Add the relevant tags to the set.
        tagset.insert(_bstr_t("TD"));
//      tagset.insert(_bstr_t("FORM"));
        tagset.insert(_bstr_t("P"));
        tagset.insert(_bstr_t("LI"));
        tagset.insert(_bstr_t("UL"));
}


TAG_BSTR_SET RelevantTags::tagset;
```

```
bool RelevantTags::isInitialized = false;

*/
```


GANZLAW, PC
PO Box 10105
Portland, Oregon 97296
Phone: (503) 228-3641
Docket No.: BIZ/01-0003
Express Mail No: EL627039307US, Deposited June 12, 2001

HTMLParser.cpp

```
#include <stdafx.h>
#include "HTMLParser.h"
#include "bizratestring.h"
#include "StringTokenizer.h"

using namespace std;


const char *HTMLParser::DISCARDED_HTML_DELIMITERS = " \n\t";

const char *HTMLParser::INCLUDED_HTML_DELIMITERS = "<>";

const string HTMLParser::EXCLUDED_BEGIN_TAGS[] = {
                "<textarea","<a", "<form", "<option" ,  "<object" ,  "<embed"};

const string HTMLParser::EXCLUDED_END_TAGS[] = {
                "</textarea","</a", "</form", "</option" ,  "</object", "</embed"};


const unsigned int HTMLParser::numExcludedTags = 6;



HTMLParser::HTMLParser(const string &html) {

        Parse(html);

}


HTMLParser::~HTMLParser() {

}


const DocumentWordList *HTMLParser::GetDocumentWords() {
        return &docWords;
}


int HTMLParser::Parse(const string &theHTML) {
        StringTokenizer
tokenizer(theHTML,DISCARDED_HTML_DELIMITERS,INCLUDED_HTML_DELIMITER
S,true);
```

```
        int numTokens=tokenizer.NumTokens();
        bool skipUntilEndOfTag=false;
        int wordIndex=-1;
        int excludedTagIndex=-1;

        for(int i=0;i<numTokens;i++) {
                wordIndex++;
                const string *token=tokenizer.GetToken(i);

                // Check if we're skipping over an anchor.

                // See if we're skipping over a tag whose body should be
                // completely exluded from annotation.
                if(excludedTagIndex != -1) {
                        // We're skipping over an exluded tag and its body
                        if(IsExcludedEndTag(token,excludedTagIndex))
                                excludedTagIndex=-1;
                        continue;
                }

                if( (excludedTagIndex=IsExcludedBeginTag(token)) != -1)
                        continue;               // Skip over this tag's body.

                // Check if we're skipping over a tag.
                if(skipUntilEndOfTag) {
                        if(token->at(0)=='>')
                                skipUntilEndOfTag=false;     // Don't worry about getting
rid of the "<" -- it'll be trimmed in the code below.
                        else
                                continue;
                }

                if(token->at(0)=='<') {
                        skipUntilEndOfTag=true;
                        continue;
                }

                // Add the word to the doc.
                int leftpos,rightpos;
                GetTokenTrimIndices(token,&leftpos,&rightpos);

                if(leftpos>rightpos)
                        continue;

                string *copy;
                if( (rightpos<token->size()-1) || (leftpos>0) )
```

```
                copy=new string(token->substr(leftpos,rightpos+1));
        else
                copy=new string(*token);

        docWords.AddWord(*copy,tokenizer.GetTokenPosition(i)+leftpos);
        delete copy;
    }

    return 0;
}


void HTMLParser::GetTokenTrimIndices(const string *token,int *leftPosIndex,int
*rightPosIndex) {
    // Return the indices into token at which non-alphanumeric characters
    // at either end stop.
    // Trim non-alphanumeric characters on the right side.
    int rightpos=token->size()-1;
    while(rightpos>=0) {
        char ch=token->at(rightpos);
        if( (!::isLetter(ch)) && (!::isNumber(ch)) )
            rightpos--;
        else
            break;
    }

    // Trim non-alphanumeric characters on the left side.
    int leftpos=0;
    while(leftpos<=rightpos) {
        char ch=token->at(leftpos);
        if( (!::isLetter(ch)) && (!::isNumber(ch)) )
            leftpos++;
        else
            break;
    }

    *leftPosIndex=leftpos;
    *rightPosIndex=rightpos;
}

int HTMLParser::IsExcludedBeginTag(const string *token) {

  int rightpo=token->size()-1;

    // checks character by character for a matching tag ;
```

```
        for(int i=0;i<numExcludedTags;i++) {
                const string *val =  &EXCLUDED_BEGIN_TAGS[i] ;
        int leftpo=0;
                if ( val->size() == token->size() ) {
                        while(leftpo<=rightpo) {
                                char ch=token->at(leftpo);
                                char ch1=val->at(leftpo);

                                if ( (ch1 == ch || abs(ch -ch1) == 32 ) )
                                        leftpo++;
                                else
                                        break ;
                        }
                if ( leftpo == rightpo + 1 )
                                return i;
                }


                }
                return -1;
}


bool HTMLParser::IsExcludedEndTag(const string *token,int tagIndex) {

        // checks character by character for a matching tag ;
   int rightpo=token->size()-1;
        const string *val =  &EXCLUDED_END_TAGS[tagIndex] ;
      int leftpo=0;

                while(leftpo<=rightpo) {
                                char ch=token->at(leftpo);
                                char ch1=val->at(leftpo);

                                if ( (ch1 == ch || abs(ch -ch1) == 32 ) )
                                        leftpo++;
                                else
                                        break ;
                        }
                if ( leftpo == rightpo + 1 )
                                return true;


        //      if(*token==EXCLUDED_BEGIN_TAGS[i])
        //      return i;

                return false;
```

GANZLAW, PC
PO Box 10105
Portland, Oregon 97296
Phone: (503) 228-3641

```
        //return tolowercopy(token)==tolowercopy(&EXCLUDED_END_TAGS[tagIndex]);
}
```

IEDocUtils.cpp

```
#include <stdafx.h>
#include "IEDocUtils.h"
#include <ExDisp.h>
#include "bstrutil.h"
#include "eBScriptUtil.h"
using namespace std;

// Function prototypes.
void   AddDocumentAndDescendents(IHTMLDocument2   *doc,DOCUMENT_VECTOR
*vec);


IHTMLDocument2 *IEDocUtils::GetMainDocument(IWebBrowser2 *browser) {
        // Return the main document of the web browser.
        IHTMLDocument2 *doc=NULL;
        IDispatch *dispatch;

        HRESULT result=browser->get_Document(&dispatch);
        if(SUCCEEDED(result)) {
                result=dispatch->QueryInterface(IID_IHTMLDocument2,(void **) &doc);
                dispatch->Release();
        }

        return doc;
}

char *IEDocUtils::GetURL(IHTMLDocument2 *doc) {
        char *theURL = NULL;

        BSTR urlBSTR;
        HRESULT result=doc->get_URL(&urlBSTR);
        if(SUCCEEDED(result)) {
                theURL=getCharArrayFromBSTR(urlBSTR);
                ::SysFreeString(urlBSTR);
        }

        return theURL;
}


char *IEDocUtils::GetHTML(IHTMLDocument2 *doc) {
        // Return the HTML of the document.
        char *html = NULL;
```

```
        IHTMLElement *body;
        HRESULT result=doc->get_body(&body);

        if(SUCCEEDED(result)) {
            html=IEDocUtils::GetHTML(body);
            body->Release();
        }

        return html;
}


char *IEDocUtils::GetHTML(IHTMLElement *elem) {
        // Get the HTML of the specified element.  This method returns NULL
        // if there is an error.  Destroy the character array that is returned
        // when you're finished with it.
        char *theHTML=NULL;
        BSTR htmlBSTR;
        HRESULT result=elem->get_innerHTML(&htmlBSTR);

        if( (SUCCEEDED(result)) && (htmlBSTR != NULL) ) {
            theHTML=getCharArrayFromBSTR(htmlBSTR);
            ::SysFreeString(htmlBSTR);
        }

        return theHTML;
}


int IEDocUtils::WalkThroughDocument(HTMLElementProcessor
*processor,IHTMLDocument2 *doc, CComPtr<IWebBrowser2> pIWebBrowser,
IDispatch* pDisp) {
        // Go through every element in the document and run the specified
        // processor on it.
        WriteLog("IEDocUtils::WalkThroughDocument: in\n");

        int retval=0;
    bool bAddSpanTag=false;
        IHTMLElementCollection *collection;
        HRESULT result=doc->get_all(&collection);
        if(SUCCEEDED(result)) {
            long numItems;
            result=collection->get_length(&numItems);

            for(long i=0;i<numItems;i++) {
                // Iterate through each element and send it to the processor.
                VARIANT nameVar;
```

```
                    nameVar.vt=VT_I4;
                    nameVar.lVal=i;

                    VARIANT emptyVar = { 0 };

                    IDispatch *dispatch;
                    result=collection->item(nameVar,emptyVar,&dispatch);


                    if( (SUCCEEDED(result)) && (dispatch != NULL) ) {
                        IHTMLElement *elem = NULL;
                        result=dispatch->QueryInterface(IID_IHTMLElement,(void
**) &elem);
                        if(SUCCEEDED(result)) {
                            processor->Process(pIWebBrowser, pDisp, elem,
bAddSpanTag);
                            elem->Release();
                        }

                        dispatch->Release();
                    }
                }

                collection->Release();
                if(bAddSpanTag)
                {
                    CComPtr<IWebBrowser2> pmainWebBrowser;
                    if(pDisp)
                        pDisp->QueryInterface(IID_IWebBrowser2,(void
**)&pmainWebBrowser);
                    CComPtr<IHTMLDocument2> pdoc;
                    if(pmainWebBrowser)
                    {
                        CComPtr<IDispatch> pdispDoc;
                        pmainWebBrowser->get_Document(&pdispDoc);
                        pdispDoc->QueryInterface(IID_IHTMLDocument2,(void
**)&pdoc);
                    }
                    if(pdoc)
                    {
                        CComVariant variantList[1];
                        variantList[0].vt = VT_DISPATCH;
                        pdoc->QueryInterface(IID_IDispatch,(void
**)&(variantList[0].pdispVal));
                        CComVariant vResult;
                        eBScript::ExecFunc( pIWebBrowser, L"ShowBuyLitesTab",
1, variantList, &vResult );
```

```
                    }
                }


        } else
                retval=-1;

        return retval;
}

int IEDocUtils::SetHTML(IHTMLElement *elem,const char *html) {
        int retval=0;

//      if( NULL != strstr(html,"<OBJECT") || NULL != strstr(html,"<IFRAME"))
//              return retval;

        BSTR htmlBSTR=makeBSTR(html);
        elem->put_innerHTML(htmlBSTR);
        ::SysFreeString(htmlBSTR);

        return retval;
}

DOCUMENT_VECTOR *IEDocUtils::GetFrames(IWebBrowser2 *browser) {
        // Returns a vector of smart points to IHTMLDocument2 objects, each of which
        // is a frame in the web browser.
        DOCUMENT_VECTOR *vec=new DOCUMENT_VECTOR;

        IHTMLDocument2 *doc=GetMainDocument(browser);
        if(doc != NULL) {
                AddDocumentAndDescendents(doc,vec);
                doc->Release();
        }

        return vec;
}

DOCUMENT_VECTOR *IEDocUtils::GetAllDescendentFrames(IHTMLDocument2 *doc)
{
        // Returns a vector of smart points to IHTMLDocument2 objects, each of which
        // is a frame under the document doc.
        DOCUMENT_VECTOR *vec=new DOCUMENT_VECTOR;
        AddDocumentAndDescendents(doc,vec);
```

```
        return vec;
}


static void AddDocumentAndDescendents(IHTMLDocument2 *doc,DOCUMENT_VECTOR *vec) {
        // The method's client should call doc->Release() when this method returns.
        IHTMLDocument2Ptr docSmartPtr(doc);          // AddRef automatically called internally.
        vec->push_back(docSmartPtr);

        IHTMLFramesCollection2 *framesCollection;
        HRESULT result=doc->get_frames(&framesCollection);
        if(SUCCEEDED(result)) {
                long numFrames;
                framesCollection->get_length(&numFrames);

                for(long l=0;l<numFrames;l++) {
                        VARIANT indexVariant;
                        indexVariant.lVal=l;
                        indexVariant.vt=VT_I4;

                        VARIANT frameVariant;

                        result=framesCollection->item(&indexVariant,&frameVariant);
                        if(SUCCEEDED(result)) {
                                IDispatch *dispatch=frameVariant.pdispVal;
                                IHTMLWindow2 *frameWindow;
                                result=dispatch->QueryInterface(IID_IHTMLWindow2,(void **) &frameWindow);

                                if(SUCCEEDED(result)) {
                                        IHTMLDocument2 *frameDoc;
                                        result=frameWindow->get_document(&frameDoc);

                                        if(SUCCEEDED(result)) {

        AddDocumentAndDescendents(frameDoc,vec);     // Add this child and its descendents.
                                                frameDoc->Release();
                                        }

                                        frameWindow->Release();
                                }

                                dispatch->Release();
                        }
```

GANZLAW, PC
PO Box 10105
Portland, Oregon 97296
Phone: (503) 228-3641
Docket No.: BIZ/01-0003
Express Mail No: EL627039307US, Deposited June 12, 2001

```
            }

            framesCollection->Release();
        }
5   }
```

ProperNounDB.cpp

```
#include <stdafx.h>
#include "ProperNounDB.h"
#include "StringTokenizer.h"
#include <iostream>
#include <fstream>
#include <strstream>
#include <assert.h>

using namespace std;


ProperNounDB::ProperNounDB() {

}


ProperNounDB::~ProperNounDB() {

}


void ProperNounDB::AddProperNoun(const ProperNoun &properNoun) {
        // Store a copy in the database.
        properNouns.push_back(properNoun);
}

const ProperNoun *ProperNounDB::GetProperNounByPhrase(string &phrase) const {
        // Return the proper noun matching the specified phrase, or return NULL
        // if no such proper noun exists in the database.
        PROPERNOUN_VECTOR::const_iterator iter=properNouns.begin();
        PROPERNOUN_VECTOR::const_iterator theEnd=properNouns.end();

        const ProperNoun *pn = NULL;
        while( (iter != theEnd) && (pn==NULL) ) {
                const ProperNoun &thePn=*iter;
                string *str=thePn.GetProperNounPhrase();
                if(*str==phrase)
                        pn=&thePn;

                delete str;
                iter++;
        }
```

```
        return pn;
}


const ProperNoun *ProperNounDB::GetProperNounById(int pnId) const {
        // Return the proper noun matching the specified proper noun id, or
        // return NULL if no such proper noun exists in the database.
        PROPERNOUN_VECTOR::const_iterator iter=properNouns.begin();
        PROPERNOUN_VECTOR::const_iterator theEnd=properNouns.end();

        const ProperNoun *pn = NULL;
        while( (iter != theEnd) && (pn==NULL) ) {
                const ProperNoun &thePn=*iter;
                if(thePn.GetId()==pnId)
                        pn=&thePn;

                iter++;
        }

        return pn;
}


const PROPERNOUN_VECTOR *ProperNounDB::GetAllProperNouns() const {
        return &properNouns;
}


int ProperNounDB::PersistToFile(const char *filename) const {
        // Write the database to a file.  It can be loaded later.
        ofstream out(filename);

        char *encodedStr=CreateStringForDB();

        out << encodedStr;

        delete[] encodedStr;

        out.flush();
        out.close();

        return 0;
}

int ProperNounDB::LoadFromFile(const char *filename) {
        ifstream in(filename);
```

```
        string completeDB;
        string str;
        while(getline(in,str)) {
                completeDB.append("\n");
                completeDB.append(str);
        }

        in.close();

        return CreateFromString(completeDB.c_str());
}


int ProperNounDB::CreateFromString(const char *encodedStr) {
        string rawData(encodedStr);

        // First, tokenize into lines containing proper nouns.  Each is on
        // its own line.  Lines are separated by a single newline character.
        StringTokenizer tokenizer(rawData,"\n","",false);
        int numProperNouns=tokenizer.NumTokens();
        for(int i=0;i<numProperNouns;i++) {
                const string *properNounLine=tokenizer.GetToken(i);
                StringTokenizer lineTokenizer(*properNounLine,"|","",false);
                int numColumns=lineTokenizer.NumTokens();

                assert(numColumns > 1);

                // First column is the proper noun phrase.  Subsequent columns
                // are proper noun ids.
                const string *properNounPhrase=lineTokenizer.GetToken(0);
                const string *properNounId=lineTokenizer.GetToken(1);

                int pnId=atoi(properNounId->c_str());
                AddProperNoun(ProperNoun(*properNounPhrase,pnId));
        }

        return 0;
}

char *ProperNounDB::CreateStringForDB() const {
        ostrstream oss;
        PROPERNOUN_VECTOR::const_iterator iter=properNouns.begin();
        PROPERNOUN_VECTOR::const_iterator theEnd=properNouns.end();

        while(iter != theEnd) {
```

```
                const ProperNoun &pn=*iter;

                // Write the proper noun phrase.
                string *pnPhrase=pn.GetProperNounPhrase();
                oss << (*pnPhrase) << "|";
                delete pnPhrase;

                // Write the proper noun id.
                oss << pn.GetId();

                // Newline line delimiter.
                oss << "\n";

                iter++;
        }

        oss << ends;

        return oss.str();
}
```

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

☐ Page(s)_____of_____ were not present for scanning. (Document title)

☐ Page(s)_____of_____were not present for scanning. (Document title)

☐ ***Scanned copy is best available.*** Drawings.